Amendments to the Specification:

Please replace the paragraph starting on page 1, line 14, with the following amended paragraph:

This application ~~is a continuation of~~ claims priority to U.S. Provisional Application No. 60/213,496 filed Jun. 23, 2000, incorporated herein by reference.

Please replace the four consecutive paragraphs starting on page 12, line 25, with the following four amended paragraphs, respectively:

FIG. 44A is an edge that asserts the value ~~true~~ false at its head and is responsive to the value true at its tail.

FIG. 44B is an edge that asserts the value ~~true~~ false at its head and is responsive to the value false at its tail.

FIG. 44C is an edge that asserts the value ~~false~~ true at its head and is responsive to the value true at its tail.

FIG. 44D is an edge that asserts the value ~~false~~ true at its head and is responsive to the value false at its tail.

Please replace the paragraph starting on page 87, line 3, with the following amended paragraph:

A static control graph (SCG) is a graph-theoretic representation of all pure control constraints. FIG. 43 shows a simple SCG. It is a bi-partite digraph, having

two types of nodes: conjunctive nodes ~~4300~~ 4316, 4330, and 4332, which, as the name implies, produce results only when all incident edges ~~4302~~ 4310 are satisfied, and disjunctive nodes 4300, 4302, 4304, 4306, and 4308, which produce results if any incident edge ~~4302~~ 4310 is satisfied. Disjunctive nodes 4300, 4302, 4304, 4306, and 4308 correspond to modes 102 in components 100 and coordinators 410 (as previously shown in FIG. 1 and FIG. 4) throughout the system. An SCG for a complete system simultaneously represents all control constraints.

Please replace the paragraph starting on page 87, line 12, with the following amended paragraph:

An SCG is a triple, G=(C, D, E), in which:

- C is a set of conjunctive nodes ~~4300~~.
- D is a set of disjunctive nodes ~~4304, 4306, and 4308~~.
- $E \subseteq [\ \{T_f, T_t\} \times \{H_f, H_t\} \times ((C \times D) \cup (D \times C))\ ]$ is a set of directed, labeled edges ~~4302~~. Edges are sensitive to either a false value or a true value at their tail ~~4310~~ 4311 (T_f or T_t) and enforce either a false value or a true value at their head 4312 (H_f or H_t). These are represented visually by a bubble at the appropriate end for a false value or the lack of a bubble for a true value.

Please replace the paragraph starting on page 87, line 21, with the following amended paragraph:

An edge ~~4302~~ 4310 in an SCG can be either enabled or disabled; it produces the value true or the value false. FIG. 44 illustrates a graphic notation for edge labels. Edges 4400 and 4402 marked with a bubble on head 4312, as in FIG. 44A and FIG. 44B, assert the value false when activated. When there is no mark on head 4312, as in FIG. 44C and FIG. 44D, an edge 4404 and 4406 asserts the value

true when activated. A bubble on tail ~~4310~~ 4311, as in FIG. 44B and FIG. 44D, indicates that edge 4406 and 4402 is sensitive to false on the node it exits. The lack of such bubbles, as in FIG. 44A and FIG. 44C, indicates that edge 4404 and 4400 is sensitive to true.

Please replace the paragraph starting on page 88, line 1, with the following amended paragraph:

Referring back to FIG. 43, the figure shows a simple SCG in which a node d 4306 must be active whenever a conjunction (a ^ b ^ c) ~~4314~~ 4332 is active and inactive whenever a node e 4308 is active. Although this looks similar to a Boolean network, it differs because the SCG edges represent implication, not connection. This is illustrated in FIG. 45, which shows a Boolean network OR node 4500; when all inputs and outputs are negated, it is equivalent to an AND node 4502 (by DeMorgan's). A disjunctive SCG node with all inputs and outputs negated 4504 is equivalent to a disjunctive node with no inputs and outputs negated 4506.

Please replace the Abstract starting on page 127, line 10, with the following amended Abstract:

An initial control graph representation of a software system may be temporally unrolled to express a potential next state of the software system. The temporally unrolled control graph may be used to generate a binary decision diagram of the software system. The binary decision diagram may be analyzed with static error checking techniques to identify unexpected behavior of the software system without fully elaborating the state space of the software system.

Amendments to the Claims:

This listing of claims will replace all prior version, and listings, of claims in the application:

Listing of Claims:

1-20. (cancelled)

21. (currently amended) A method for converting a control graph representation of a software system, having a state space and an initial state, into a binary decision diagram of the software system, the control graph and the binary decision diagram to reside at least partially in memory of a computer, the method comprising:

transforming the control graph to express a potential next state of the software system ~~after a predetermined period of time~~, the transforming performed by the computer; and

generating a binary decision diagram based on the transformed control graph, whereby ~~known~~ static error checking ~~techniques~~ may be used to ~~further~~ identify ~~any~~ unexpected behavior of the software system without ~~incurring the cost of~~ fully elaborating the state space of the software system.

22. (currently amended) A method according to claim 21 wherein transforming the control graph comprises temporally unrolling the control graph.

23. (currently amended) A method according to claim 22 wherein generating the binary decision diagram comprises using ~~the~~ an apply algorithm on a characteristic function of the temporally unrolled control graph.

24. (currently amended) A method according to ~~claim 23~~ claim 22 wherein temporally unrolling the control graph comprises:

creating a copy of each disjunctive node, each disjunctive node represents a boolean guard on a functional object within one of the software elements;

creating a copy of each conjunctive node, each conjunctive node represents a conjunctive boolean guard on state changes within the software system;

creating a copy of each action node, each action node represents a functional object within one of the software elements that is responsive to a control interaction and capable of producing a control interaction, if the functional object ~~it~~ the action node represents performs a predetermined function without a predetermined delay;

for each delayed action node which represents a functional object within one of the software elements that has a predetermined delay in responding to or ~~producing,~~ producing a control interaction, (a) creating a sensing edge to connect the delayed action node to a corresponding node in the control graph representing the initial state of the ~~system~~ system, and (b) creating an outgoing edge to connect the corresponding ~~node,~~ node in the control graph representing the initial state of the system, to a corresponding next ~~node,~~ node which ~~represent~~ represents the potential next state of the system;

for outgoing ~~edge,~~ edge that is also an event edge, connecting the outgoing edge to a create event disjunctive ~~node,~~ node which represents an event generated by the corresponding node in the control graph representing the initial state of the system;

for each created event disjunctive node, creating an edge from the created event disjunctive node to an event conjunctive node; and

for each event conjunctive node, creating an edge from the node that generated the event to the event conjunctive node, and creating an edge from the event conjunctive node to the copy of the node that generated the event.

25. (cancelled)

26. (new) A method comprising:

transforming a first control graph representation of a software system into a second control graph representation of the software system, wherein the software system comprises a state space that changes over time, and wherein the first control graph representation expresses the software system at an initial state and the second control graph representation expresses the software system at a potential next state; and

generating a binary decision diagram based at least in part on the second control graph;

wherein the binary decision diagram is suitable for use with static error checking techniques to identify unexpected behavior of the software system without fully elaborating the state space of the software system;

wherein the first control graph representation of the software system resides at least partially within memory of a computer, and

wherein the operation of transforming the first control graph representation into the second control graph representation is performed by the computer.